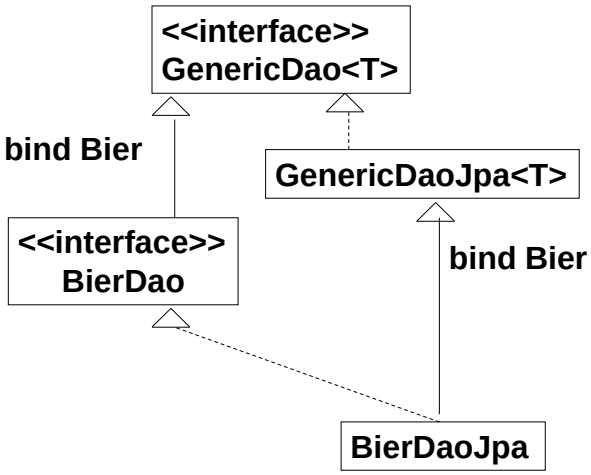
Simplified DAO Pattern with Generics



1

Simplified DAO Pattern with Generics

```
package repository;
import java.util.List;

public interface GenericDao<T> {

   public List<T> findAll();
   public T get(Long id);
   public T update(T object);
   public void delete(T object);
   public void insert(T object);
   public boolean exists(Long id);

}
```

2

Simplified DAO Pattern with Generics


```
package repository;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class GenericDaoJpa<T> implements GenericDao<T> {

   private static final String PU_NAME = "bierWinkels";
   private static final EntityManagerFactory emf =
         Persistence.createEntityManagerFactory(PU_NAME);
         protected static final EntityManager em =
                                    emf.createEntityManager();
   private final Class<T> type;

   public GenericDaoJpa(Class<T> type) {
      this.type = type;
   }
```

3

GenericDaoJpa<T>

```
   public static void closePersistency() {
      em.close();
      emf.close();
   }

   public static void startTransaction() {
      em.getTransaction().begin();
   }

   public static void commitTransaction() {
      em.getTransaction().commit();
   }

   public static void rollbackTransaction() {
      em.getTransaction().rollback();
   }
```

4

GenericDaoJpa<T>

```
@Override
public List<T> findAll() {
    return em.createQuery("select entity from " +
        type.getName() + " entity", type).getResultList();
    /*return em.createNamedQuery(type.getName()+
        ".findAll", type).getResultList();*/
}

@Override
public T get(Long id) {
    T entity = em.find(type, id);
    return entity;
}

@Override
public T update(T object) {
    return em.merge(object);
}
```

5

GenericDaoJpa<T>

```
@Override
public void delete(T object) {
    em.remove(em.merge(object));
}

@Override
public void insert(T object) {
    em.persist(object);
}

@Override
public boolean exists(Long id) {
    T entity = em.find(type, id);
    return entity != null;
}

}
```

6

Simplified DAO Pattern with Generics

```
package repository;

import domain.Bier;
import javax.persistence.EntityNotFoundException;

public interface BierDao extends GenericDao<Bier>  {

      public Bier getBierByName(String name)
                        throws EntityNotFoundException;
}
```

7

Simplified DAO Pattern with Generics

```
package repository;

import domain.Bier;
import javax.persistence.EntityNotFoundException;
import javax.persistence.NoResultException;

public class BierDaoJpa extends GenericDaoJpa<Bier>
                              implements BierDao {

   public BierDaoJpa() {
      super(Bier.class);
   }
```

8

Simplified DAO Pattern with Generics

```
@Override
public Bier getBierByName(String name)
        throws EntityNotFoundException {
    try {
        return em.createNamedQuery("Bier.findByName", Bier.class)
                .setParameter("bierNaam", name)
                .getSingleResult();
    } catch (NoResultException ex) {
        throw new EntityNotFoundException();}
  }
}
```

```
@Entity
@NamedQueries({
    @NamedQuery(name = "Bier.findByName",
                query = "select b from Bier b where b.naam = :bierNaam")
})
public class Bier implements Serializable {
```

9

Use DAO

```
public class DomeinController {

    private GenericDao<Winkel> winkelRepo;
    private BierDao  bierRepo;

…

   //IN CONSTRUCTOR:
    setWinkelRepo(new GenericDaoJpa<>(Winkel.class));
    setBierRepo(new BierDaoJpa());

…

  //SETTER-INJECTIONS
    public void setWinkelRepo(GenericDao<Winkel> mock){
        winkelRepo = mock;
    }
    public void setBierRepo(BierDao mock){
        bierRepo = mock;
    }
```

10

Use DAO

```
…
public void voegBierBijWinkel(String bierNaam, String winkelNaam)
                                throws IllegalArgumentException {
    Optional<Winkel> winkel = getWinkelList().stream()
          .filter( w -> w.getNaam().equalsIgnoreCase(winkelNaam))
          .findFirst();
    if (!winkel.isPresent()) {
          throw new IllegalArgumentException("winkel " +
                                    winkelNaam + " komt niet voor");
    }
    Bier bier;
    try {
        bier = bierRepo.getBierByName(bierNaam);
    } catch(EntityNotFoundException ex) {
       throw new IllegalArgumentException("bier " + bierNaam +
                                                  " komt niet voor");
    }
    GenericDaoJpa.startTransaction();
    winkel.get().addBier(bier);
    GenericDaoJpa.commitTransaction();
  }
```


Use DAO

```
…
    private List<Winkel> getWinkelList(){
        if (winkelList==null){
                winkelList=winkelRepo.findAll();
        }
        return winkelList;
    }

…

    public void close() {
        GenericDaoJpa.closePersistency();
    }
…
```

```
[Op den hoek, Putteke]
```

Testklasse

```
public class DomeinTest {

   private GenericDao<Winkel> winkelRepo;

   private BierDao  bierRepo;

   private DomeinController domein;…

   @Before
   public void before() {
      domein = new DomeinController(false);
      winkelRepo = Mockito.mock(GenericDao.class);
      bierRepo = Mockito.mock(BierDao.class);
      domein.setBierRepo(bierRepo);
      domein.setWinkelRepo(winkelRepo);
   }

   @Test
   public void voegBierBijWinkel() { …
```

13